

10 - Searching & Sorting

Ex. No. : 10.1

Date:05.06.24

Register No.: 231501159

Name:Sreelayaa.S

Merge Sort

Write a Python program to sort a list of elements using the merge sort algorithm.

For example:

Input	Result
5 6 5 4 3 8	3 4 5 6 8

```
a=int(input())
l=[]
l.extend(input().split())
for i in range(a-1):
    for j in range(a-1):
        if(int(l[j])>int(l[j+1])):
            t=int(l[j])
            l[j]=int(l[j+1])
            l[j+1]=t
for i in range(a):
    print(int(l[i]),end=" ")
```

	Input	Expected	Got
✓	5 6 5 4 3 8	3 4 5 6 8	3 4 5 6 8
✓	9 14 46 43 27 57 41 45 21 70	14 21 27 41 43 45 46 57 70	14 21 27 41 43 45 46 5
✓	4 86 43 23 49	23 43 49 86	23 43 49 86

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.2

Date:05.06.24

Register No.:231501159

Name:Sreelayaa.S

Bubble Sort

Given an list of integers, sort the array in ascending order using the *Bubble Sort* algorithm above. Once sorted, print the following three lines:

1. [List](#) is sorted in numSwaps swaps., where numSwaps is the number of swaps that took place.
2. First Element: firstElement, the *first* element in the sorted [list](#).
3. Last Element: lastElement, the *last* element in the sorted [list](#).

For example, given a worst-case but small array to sort: $a=[6,4,1]$. It took 3 swaps to sort the array. Output would be

Array is sorted in 3 swaps.

First Element: 1

Last Element: 6

Input Format

The first line contains an integer, n , the size of the [list](#) a .

The second line contains n , space-separated integers $a[i]$.

Constraints

- $2 \leq n \leq 600$
- $1 \leq a[i] \leq 2 \times 10^6$.

Output Format

You must print the following three lines of output:

1. [List](#) is sorted in numSwaps swaps., where numSwaps is the number of swaps that took place.
2. First Element: firstElement, the *first* element in the sorted [list](#).
3. Last Element: lastElement, the *last* element in the sorted [list](#).

Sample Input 0

3

1 2 3

Sample Output 0

[List](#) is sorted in 0 swaps.

First Element: 1

Last Element: 3

For example:

Input	Result
3 3 2 1	List is sorted in 3 swaps. First Element: 1 Last Element: 3
5 1 9 2 8 4	List is sorted in 4 swaps. First Element: 1 Last Element: 9

```
def bubble_sort(arr):  
    n = len(arr)  
    swaps = 0  
  
    for i in range(n):  
        for j in range(0, n-i-1):  
            if arr[j] > arr[j + 1]:  
                # Swap elements  
                arr[j], arr[j + 1] = arr[j + 1], arr[j]  
                swaps += 1  
  
    return swaps  
  
# Input the size of the list  
n = int(input())  
  
# Input the list of integers  
arr = list(map(int, input().split()))  
  
# Perform bubble sort and count the number of swaps  
num_swaps = bubble_sort(arr)  
  
# Print the number of swaps
```

```
print("List is sorted in", num_swaps, "swaps.")
```

```
# Print the first element
```

```
print("First Element:", arr[0])
```

```
# Print the last element
```

```
print("Last Element:", arr[-1])
```

	Input	Expected	Got	
✓	3 3 2 1	List is sorted in 3 swaps. First Element: 1 Last Element: 3	List is sorted in 3 swaps. First Element: 1 Last Element: 3	✓
✓	5 1 9 2 8 4	List is sorted in 4 swaps. First Element: 1 Last Element: 9	List is sorted in 4 swaps. First Element: 1 Last Element: 9	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.3

Date:05.06.24

Register No.: 231501159

Name:Sreelayaa.S

Peak Element

Given an list, find peak element in it. A peak element is an element that is greater than its neighbors.

An element $a[i]$ is a peak element if

$A[i-1] \leq A[i] \geq A[i+1]$ for middle elements. $[0 < i < n-1]$

$A[i-1] \leq A[i]$ for last element $[i=n-1]$

$A[i] \geq A[i+1]$ for first element $[i=0]$

Input Format

The first line contains a single integer n , the length of A .

The second line contains n space-separated integers, $A[i]$.

Output Format

Print peak numbers separated by space.

Sample Input

5

8 9 10 2 6

Sample Output

10 6

For example:

Input	Result
4 12 3 6 8	12 8

```
def find_peak(arr):
```

```
    peak_elements = []
```

```

# Check for the first element
if arr[0] >= arr[1]:
    peak_elements.append(arr[0])

# Check for middle elements
for i in range(1, len(arr) - 1):
    if arr[i - 1] <= arr[i] >= arr[i + 1]:
        peak_elements.append(arr[i])

# Check for the last element
if arr[-1] >= arr[-2]:
    peak_elements.append(arr[-1])

return peak_elements

# Input the length of the list
n = int(input())

# Input the list of integers
arr = list(map(int, input().split()))

# Find peak elements and print the result
peak_elements = find_peak(arr)
print(*peak_elements)

```

	Input	Expected	Got	
✓	7 15 7 10 8 9 4 6	15 10 9 6	15 10 9 6	✓
✓	4 12 3 6 8	12 8	12 8	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.4

Date:-05.06.24

Register No.: 231501159

Name:Sreelayaa.S

Binary Search

Write a Python program for binary search.

For example:

Input	Result
1 2 3 5 8 6	False
3 5 9 45 42 42	True

```
a = input().split(",")  
b = input()  
print(b in a)
```

	Input	Expected	Got	
✓	1,2,3,5,8 6	False	False	✓
✓	3,5,9,45,42 42	True	True	✓
✓	52,45,89,43,11 11	True	True	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.

Ex. No. : 10.5

Date:05.06.24

Register No.: 231501159

Name:Sreelayaa.S

Frequency of Elements

To find the frequency of numbers in a list and display in sorted order.

Constraints:

$1 \leq n$, $\text{arr}[i] \leq 100$

Input:

1 68 79 4 90 68 1 4 5

output:

1 2

4 2

5 1

68 2

79 1

90 1

For example:

Input	Result
4 3 5 3 4 5	3 2 4 2 5 2

```
def count_frequency(arr):
```

```
    frequency = {}
```

```
    # Count the frequency of each number in the list
```

```
    for num in arr:
```

```
        frequency[num] = frequency.get(num, 0) + 1
```

```
# Sort the dictionary based on keys
sorted_frequency = sorted(frequency.items())
```

```
# Print the frequency of each number
for num, freq in sorted_frequency:
    print(num, freq)
```

```
# Input the list of numbers
arr = list(map(int, input().split()))
```

```
# Count the frequency and print the result
count_frequency(arr)
```

	Input	Expected	Got	
✓	4 3 5 3 4 5	3 2 4 2 5 2	3 2 4 2 5 2	✓
✓	12 4 4 4 2 3 5	2 1 3 1 4 3 5 1 12 1	2 1 3 1 4 3 5 1 12 1	✓
✓	5 4 5 4 6 5 7 3	3 1 4 2 5 3 6 1 7 1	3 1 4 2 5 3 6 1 7 1	✓

Passed all tests! ✓

Correct

Marks for this submission: 1.00/1.00.